REMARKS

Claims 1-8, 10-12, 15-20, 22-25, 27, 30-39, 42-43, and 45-50 are pending. Claims 1-8, 10-12, 15-20, 22-25, 27, 30-39, 42-43, and 47-50 are allowed. Claims 45-46 stand rejected under 35 U.S.C. § 101 as being directed to non-statutory subject matter. Claim 45 stands rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,039,346 to Ratnakar. Claim 46 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 6,038,346 to Ratnakar in view of U.S. Patent No. 6,512,793 to Maeda.

Reconsideration is requested. No new matter is added. Claim 45 is amended. The rejections are traversed. Claims 1-8, 10-12, 15-20, 22-25, 27, 30-39, 42-43, and 45-50 remain in the case for consideration.

The Examiner indicated that he did not receive a copy of the Appendix. The Applicant does not understand what happened, because the Applicant has a record of sending the Appendix. In any event, a new copy of the Appendix is hereby provided, along with a copy of the return postcard from the U.S. Patent & Trademark Office, showing the original submission of the Appendix.

REJECTION OF CLAIMS UNDER 35 U.S.C. § 101

The Examiner rejected claims 45-46 as being directed toward nonstatutory subject matter under MPEP 2106 IV B.1(b). The Examiner takes the position that claims 45-46 do not exhibit any functional interrelationship with the way in which computing processes are performed. But like the claims in *In re Lowry*, 32 U.S.P.Q. 1031 (Fed. Cir. 1994), the memory of claim 45 does perform a function. Claim 45 is directed toward a memory having a structure that enables an application to manage data relating to compression of an image.

The Examiner stated that the Court of Appeals for the Federal Circuit reversed the decision of the Board of Patent Appeals and Interferences, and therefore the Board's reasoning is inapplicable. The Examiner is incorrect. The procedural history of Lowry was that the Examiner had rejected the claims under 35 U.S.C. §§ 101, 102, and 103. On Appeal, the Board of Patent Appeals and Interferences reversed the Examiner's rejection under 35 U.S.C. § 101, while sustaining the rejection under 35 U.S.C. §§ 102 and 103. Lowry appealed the rejection under 35 U.S.C. § 102 and 103, and the Court of Appeals for the Federal Circuit reversed the rejection under 35 U.S.C. §§ 102 and 103.

In the opinion of the Court of Appeals, "[t]he Board found that claims 1 through 5 [of Lowry], directed to a memory containing stored information, as a whole, recited an article of manufacture. The Board concluded that the invention claimed in claims 1 through 5 was

statutory subject matter" (*In re Lowry*, 32 U.S.P.Q.2d 1031, 1033 (Fed. Cir. 1994)). Because the Court of Appeals reviewed the case *de novo*, the Court of Appeals could have reviewed the allowability of the claims under 35 U.S.C. § 101 *sua sponte*. That the Court of Appeals did not undertake such a review and held the claims to be patentable shows that the Court of Appeals agreed with the Board with respect to the patentability of the claims under 35 U.S.C. § 101. Accordingly, the Board's reasoning with respect to patentability of the claims under 35 U.S.C. § 101 is valid, and the Applicant's argument as to the applicability of the Board's argument is appropriate. Since the Applicant claims "a memory containing stored information" as in Lowry, claims 45-46 are allowable under 35 U.S.C. § 101 as patentable subject matter.

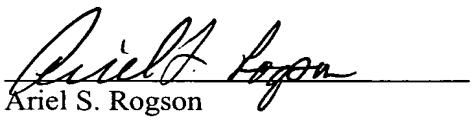
REJECTION OF CLAIMS UNDER 35 U.S.C. § 102(e)

In maintaining a rejection of claim 45, the Examiner indicated that the Applicant argued that the claimed "subset of pixels" does not exclude the possibility of the subset being the entire image. Claim 45 has been amended to explicitly exclude from the subset of the pixels a pixel from the image. Accordingly, the "subset" is now a proper subset, and is not anticipated by Ratnakar. Accordingly, claims 45-46 are now allowable.

For the foregoing reasons, reconsideration and allowance of claims 1-8, 10-12, 15-20, 22-25, 27, 30-39, 42-43, and 45-50 of the application as amended is solicited. The Examiner is encouraged to telephone the undersigned at (503) 222-3613 if it appears that an interview would be helpful in advancing the case.

Respectfully submitted,

MARGER JOHNSON & McCOLLOM, P.C.



Ariel S. Rogson
Reg. No. 43,054

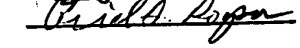
I HEREBY CERTIFY THAT THIS CORRESPONDENCE IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE AS FIRST CLASS MAIL IN AN ENVELOPE ADDRESSED TO:

☐ COMMISSIONER OF PATENTS AND TRADEMARKS WASHINGTON D.C. 20231

☒ MAIL STOP
COMMISSIONER FOR PATENTS
BOX 1450
ALEXANDRIA, VA 22313-1450

☐ BOX
COMMISSIONER
FOR TRADEMARKS 2900 CRYSTAL DRIVE ARLINGTON, VA 22202-3513

ON: 26 June 2004



MARGER JOHNSON & McCOLLOM, P.C.
1030 SW Morrison Street
Portland, OR 97205
503-222-3613
Customer No. 20575

APPENDIX A

```
/*
 * (C) Copyright 2000 by OnMercial.com, Inc.
 * All rights reserved
 *
 * General pixel encoding routine.
 *
 * The routine ac_out is a standard range encoder.  The routine UpdateModel
 * is responsible for updating the model to reflect the new probability
 * values given the color of the current pixel.
 */
int AmfCoder::encode(int a_y,int a_x) {
/*
 * pc is the index into the color palette for the current pixel.
 * pl is the index into the color palette for the left neighbor pixel.
 * plu is the index into the color palette for the upper neighbor pixel.
 * ptr is a two-dimensional array storing the indices into the color
 *   palette in the image.
 */
  int pc;
  int pl;
  int pu;
  int ptr;
/*
 * Identify the index into the color palette for the current pixel.
 */
  pc=(*frame)(a_x,a_y);
/*
 * Assert the proposition that the index into the color palette for the
 * current pixel is a valid index (i.e., not out of bounds).
 */
  assert(pc<pal_size+1);
/*
 * Identify the indices into the color palette for the upper neighbor.
 */
  if(a_y>frame->y0) {
    pu=(*frame)(a_x,a_y-1);
  } else {
/*
 * Use the index in the color palette for the transparent color to indicate
 * that the current pixel is in the top row of the image.
 */
    pu=pal_size;
  }
/*
 * Identify the indices into the color palette for the left neighbor.
 */
  if(a_x>frame->x0) {
    pl=(*frame)(a_x-1,a_y);
  } else {
/*
 * Use the index in the color palette for the transparent color to indicate
 * that the current pixel is in the top row of the image.
 */
    pl=pal_size;
  }
/*
 * Encode the current pixel.
 *
```

```
        * Mode 1: The left and upper pixels have the same color.
        */
         if (pu==pl) {
       /*
        * Determine the proper offsets into the frq_eq array for the current
        * probability model.
        */
           ptr=3*context;
       /*
        * Mode 1a: The current pixel has the same color as the left and upper
        * neighbors.
        */
           if (pl==pc) {
       /*
        * Encode the mode and update the model.
        */
             coder->ac_out(frq_eq[ptr+0],frq_eq[ptr+1],frq_eq[ptr+2]);
             UpdateModel(ptr+frq_eq,2,0,ALEVEL);
       /*
        * Select the proper model for the next pixel.
        */
             context=0;
       /*
        * Mode 1b: The current pixel has a different color than the left and upper
        * neighbors.
        */
           } else {
       /*
        * Encode the mode and the index into the color palette for the current
        * pixel, and update the model.
        */
             coder->ac_out(frq_eq[ptr+1],frq_eq[ptr+2],frq_eq[ptr+2]);
             encode_0(pl,pl,pc);
             UpdateModel(ptr+frq_eq,2,1,ALEVEL);
       /*
        * Select the proper model for the next pixel.
        */
             context=1;
           }
       /*
        * Mode 2: The left and upper pixels have different colors.
        */
         } else {
       /*
        * Determine the proper offsets into the frq_ne array for the current
        * probability model.
        */
           ptr=4*context;
       /*
        * Mode 2a: The current pixel has the same color as the left neighbor only.
        */
           if (pl == pc) {
       /*
        * Encode the mode and update the model.
        */
             coder->ac_out(frq_ne[ptr+0],frq_ne[ptr+1],frq_ne[ptr+3]);
             UpdateModel(ptr+frq_ne,3,0,ALEVEL2);
       /*
        * Select the proper model for the next pixel.
```

```
 */
      context=2;
/*
 * Mode 2b: The current pixel has the same color as the upper neighbor
 * only.
 */
   } else if (pu == pc) {
/*
 * Encode the mode and update the model.
 */
      coder->ac_out(frq_ne[ptr+1],frq_ne[ptr+2],frq_ne[ptr+3]);
      UpdateModel(ptr+frq_ne,3,1,ALEVEL2);
/*
 * Select the proper model for the next pixel.
 */
      context=3;
/*
 * Mode 2c: The current pixel has a different color than the left and upper
 * neighbors.

 */
   } else {
/*
 * Encode the mode and the index into the color palette for the current
 * pixel, and update the model.
 */
      coder->ac_out(frq_ne[ptr+2],frq_ne[ptr+3],frq_ne[ptr+3]);
      UpdateModel(ptr+frq_ne,3,2,ALEVEL2);
      encode_0(pl,pu,pc);
/*
 * Select the proper model for the next pixel.
 */
      context=4;
   }
  }
  return 0;
}


/*
 * Encoding routine for encoding the current pixel when the current pixel
 * has a color different from both the left and upper neighbors.
 */
int AmfCoder::encode_0(int a_l, int a_u, int a_c) {
/*
 * xl marks the low end of the range of probability values for a color
 *   in the color palette.
 * xh marks the high end of the range of probability values for a color
 *   in the color palette.
 * xtot stores the total number of occurrences of each color in the
 *   color palette.
 */
  U16B xl;
  U16B xh;
  U16B xtot;

  U16B i;
/*
 * Initialize the low end of the range of probability values for the first
 * color in the color palette.
```

```
 */
  xl=0;
/*
 * Calculate the total number of occurrences of each color in the color
 * palette.
 */
    xtot=frq_0[pal_size+1];
/*
 * Exclude from the total number of occurrences of each color in the color
 * palette the number of occurrences of the colors of the left and upper
 * neighbors.
 */
  xtot -= frq_0[a_l];
/*
 * Only exclude the number of occurrences of the color of the upper
 * neighbor if it is different from the color of the left neighbor (i.e.,
 * we are not in mode 1).
 */
  if (a_l != a_u) {
    xtot -= frq_0[a_u];
  }
/*
 * Scan through the colors in the color palette.
 */
  for (i=0;i<pal_size+1;i++) {
/*
 * Ignore the colors or the left and upper neighbors.
 */
    if (i==a_l || i==a_u) {
      continue;
    }
/*
 * Set the high end of the range of probability values for the current
 * color to be the low end of the range of probability values for the
 * current color plus the number of occurrences of the current color.
 */
    xh=xl+frq_0[i];
/*
 * If the current color is that of the current pixel, encode the current
 * color and update the model (including the number of occurrences of each
 * color in the color palette).
 */
    if (i==a_c) {
      coder->ac_out(xl, xh, xtot);
      UpdateModel2(frq_0,(U16B)(pal_size+1),i,ALEVEL0);
/*
 * Break out of the for-loop.
 */
      break;
    }
/*
 * Set the low end of the range of probability values for the next color to
 * be the high end of the range of probability values for the current
 * color.
 */
    xl=xh;
  }
  return 0;
}
```